ADEPT

AUTOMATED DESIGN EXPERT THEORETICAL MANUAL

Prepared by:
John P. Frazier
U.V.L. Narayana
- of -
Autodesk, Inc.
1303 Hightower Trail, Suite 170
Atlanta, GA 30350

Prepared for :
NASA - Lewis Research Center
Cleveland, OH 44135

Contract # NAS3-25150
An Expert System for Finite Element Modeling

June 20, 1990

ADEPT THEORETICAL MANUAL

TABLE OF CONTENTS

Figures

Appendices

OBJECTIVES FOR ADEPT

From the start, **Adept** was targeted to be a system that would allow a design engineer having little or no practical experience in performing finite element analysis to be able to choose an appropriate modeling and analysis strategy for a given problem. There was the original restriction that the system would be limited to structural analysis. During the development of the system it became evident that additional restrictions should be added. Early in the development of the system the NASA Lewis project manager made the wise recommendation that instead of building an expert system that attempted to handle a broad variety of capabilities it would be prudent to limit the bounds of the working environment. His experience with expert systems had shown many examples of systems that began their development cycle attempting to do all; unfortunately, these systems ended up in reality doing very little and in effect serving as a learning experience for choosing your domain carefully. After this discussion, there were additional restrictions added to the **Adept** system. The project was limited to linear static analysis of isotropic materials. Within any one analysis problem, there would not be multiple element types or multiple material properties. The element class was limited to triangles and tetrahedrals due to the choice of the automatic finite element modeler. Special analysis classes for plane stress and plane strain would be included.

Every attempt would be made to develop the system with an open architecture. Any workable system should be capable of future additions. The rule base would be targeted as a general set that could handle objects that would be loaded with forces and pressures. The open architecture would allow the user to alter the rule base in the event that application of finite element analysis to a certain discipline might have some peculiar requirements. For example, the modeling practice used in the automobile industry might be similar, but not quite the same as the modeling practice used in an aerospace application. The alteration or addition of certain rules might allow the user to cause those meshing changes to be realized. Refer to the programmer's manual for ways to customize the starting rule set of **Adept.**

One of the major intents of **Adept** was to make the job of performing finite element analysis as simple as possible for the designer. Many of the techniques fundamental to the practice of finite element analysis demand that assumptions be made. The physical world situation is simplified to a computer model that should require the minimum amount of resources to attain the desired solution. One of the goals then is to decrease the needed computation time and still supply a result that sufficiently describes the desired output. This maybe a stress contour, or a maximum stress value, or perhaps a deflected shape.

DEVELOPMENT OF ADEPT

A survey was done of the commercial or otherwise available software that would comprise the computer aided design(CAD), the finite element modeling(FEM), and the expert system(ES) portion of the **Adept** system. A study was also done of the most popular finite element analysis(FEA) codes to determine which solvers to interface with. The individual parts would be integrated into a workable system. Two major considerations in the final decision of these software components were the capabilities of the individual packages and the ability to tailor these into a system. Access to source code was an important consideration in that our development team could add the needed capabilities that were not in the standard commercial package. Access to source code also allowed us the opportunity to design a system that would be computationally efficient and would in effect have an acceptable response time for various system operations.

Refer to Appendix A for the Task 1 report, "A Survey of Relevant CAD, FEA, and Expert System Capabilities", for a full description of the software packages considered and exactly why certain programs were eventually decided upon. For a computer aided design package AutoSolid 3.1 from Autodesk was chosen. AutoSolid also included an automatic finite element modeling option which was used. The survey of existing FEA packages showed that NASTRAN from MacNeal Schwendler and ANSYS from Swanson Analysis dominate the finite element field. These were the packages we would interface with. For an expert system shell the CLIPS package from the Johnson Space Center was chosen.

Two routes were taken to build the initial expert system rule base for the **Adept** program. An in-depth literature search was performed with the goal of locating substantial information on the practice of building finite element models. Information relating to the general use of FE was located and the manuals for NASTRAN and ANSYS were studied. The information that was found on theoretical issues of FEA was only coded into the rule base if the practical consequences of such an issue could be boiled down to a rule that the program should be bound to follow. The personal experience of both members of the design team and independent engineers with practicing experience in the application of finite element analysis was used as a resource to gain rules for the system. Sample problems were setup and those interviewed were questioned as to how they would generate an applicable model for some specified real world problem. They were asked how much of the real world situation they would model, what element type they would use, how many elements they would use, and what aspects of the problem would they ignore.

The rules that were gained were then divided into two broad categories. Some rules applied to the general principles of FEA. These formed the general knowledge base. Other rules specific to using NASTRAN or ANSYS were divided into groups of package specific rules.

In the integration of the various software programs and rules into a workable expert system for finite element modeling, it was evident that there were a

few missing pieces. The largest piece that was needed we will call the environment descriptor. This is the software that was generated to make determinations about the solid that was to be analyzed. As an example, the rules might need to know if the solid was an extrusion. AutoSolid's geometric definition includes both a CSG tree and a list of surfaces, edges, and vertices and their position in space, but this does not immediately designate the solid into such a class. Software was needed to check the mathematical definition to see if the solid was an extrusion. Additional checks were needed to classify the other symmetries of the solid.

The **Adept** system was developed as a combination of pieces which would later be assembled. The rule base in a large part specified the information that was needed to classify the modeling situation. In the event that this information was not obtainable directly from the AutoSolid database (which was almost always), intermediate routines would be written for the environment descriptor to quantify this information from the geometric database. This allowed various members of the design team to independently develop and test their separate parts of the project. These pieces would later be assembled and the last few missing pieces would be supplied through a joint effort. Separating the pieces allowed a quicker development of the entire project, and at a later date the missing pieces were evident and were developed and added.

The hardware selection that had originally been proposed as Apollo workstations was changed to a Sun 4 machine and two Sun 386i machines. By waiting as late in the project as possible to specify the computer manufacturer and model, the most economical computing hardware was obtained. The ever decreasing cost of processing power makes it a wise choice to wait as long as possible to actually buy the hardware.

IMPLEMENTATION AND TESTING OF THE SYSTEM

Adept functions as an add-on application program tied to AutoSolid. This is accomplished through the AutoSolid Programming Interface (API) capability of AutoSolid. A full explanation of how to develop and run an application program tied to the geometric and windowing abilities of AutoSolid is covered in the AutoSolid Programming Interface Reference Manual. Figure 1 depicts the layout for the **Adept** system. AutoSolid can be conceptualized as the User Interface, the Solid Modeler, and the Geometric Database. The User Interface handles all the menuing, windowing, and viewing of the solid. The user builds, alters, and queries the existing solid via this interface. The Solid Modeler handles the mathematics involved in dealing with the solids. This part can be termed the geometry engine. All information defining the solid resides in the Geometric Database. Changes to the definition of the solid result in updates to the database. The development of programs with API allows the user to access routines in both the User Interface(example, new menus can be generated) and the Solid Modeler(example, commands can be given to alter the working copy of the solid).

The standard version of AutoSolid also includes a package to perform automatic finite element mesh generation for solids and surfaces. The package generates triangular and tetrahedral elements via an octree method. The user controlled package requires the user to choose the element type, the mesh density, the desired surfaces or solid, apply the loading and boundary conditions, and to perform modeling checks. **Adept** automatically accomplishes many of these operations. The standard version of AutoSolid.FEM is depicted in Figure 1 with the FEA Input Data File circle. Once the problem has been solved with a finite element analysis program the solution output can be postprocessed to view deformations and stress contours.

In Figure 1 everything to the right of the User Interface box is the **Adept** module. This is an application program that is attached to AutoSolid via the API hookup. The Environment Descriptor will query the Geometric Database and make classifications of the solid of interest. This could relate to symmetries, the presence of depressions and protrusions, the relative size of cutouts, etc. All these determinations are made from software written for **Adept**. The Expert Controller determines where required information will be obtained from. There are times when information can be algorithmically obtained and there are other times where the user must supply information. The Controller also can be viewed as the additional needed software that does not fall under the Environment Descriptor. In the way that **Adept** is designed, the loads and boundary conditions are associated directly with the solid and features of the solid. In the standard AutoSolid.FEM loads and boundary conditions are defined directly on the finite element mesh. The additional software required to supply this capability can be pictured as part of the Expert Controller.

The Inference Engine is an embedded version of CLIPS, an expert system shell from the NASA Johnson Space Flight Center. The knowledge is in the form of a rule base that drives the accumulation of needed information from

both the user and AutoSolid. Decision branches are made based on the facts in the working memory defining the specific situation. Our set of rules for finite element modeling are depicted in Figure 1 as the General Knowledge Base and the Package Specific Knowledge Bases 1 and 2(for Ansys and Nastran). Appendix B is a listing of the General Knowledge Base in a CLIPS format. It should be noticed that although many technical rules for modeling were acquired for the full rule base, the task of developing a working finite element system brought to light the need for other information to be acquired and categorized. Personal experience and the interviews with practicing experts allowed the process of building an appropriate meshing scheme to be broken in a procedure that encompassed a series of stages.

The stages for proceeding from a physical world situation to a computer model were segmented as follows. This is also the path that is followed in **Adept**. The purpose for the analysis must be defined along with the resources that are available to accomplish the analysis. It will be important to know whether a general behavior, basically a deformation study is desired or whether a detailed stress analysis is required. This difference can greatly influence the model that must be generated.

The solid, the loadings, and the boundary conditions must be defined at this first stage. Merely having the solid of interest is not enough. These are the pieces that any analyst requires in the process of deciding what to model and how to model it. Without these predescribed pieces there is almost no reason to continue the generation of a mathematical model.

Next, the geometric features of the solid must be ranked as to how they affect the stress results. Some features may be ignorable, others may require an increased mesh density in their vicinity. In the **Adept** program these features can be ranked as both simple features(surfaces, edges, and vertices) and as complex features(depressions and protrusions). Very unimportant complex features will be removed from the working copy of the original solid via boolean operations.

Then the geometric symmetries of the working solid will be categorized. Many of the time saving techniques of finite element modeling allow the solid to be reduced at mirror planes if certain conditions exist. The fact that as the model size doubles the computational requirements increase by a factor of four makes this such an important technique. Redundancies in the final result, should that be deformations or stresses should not be recalculated in an effort to save computation time.

The loads and boundary conditions(lbcs) are checked against the geometric symmetries that are present. The lbcs are categorized and the working copy of the solid is reduced along these planes of symmetry, if possible. The appropriate boundary conditions are defined in the event the solid is reduced in any direction.

Attempts are made to categorize the solid as rod-like(one dimensional), plate-like(two dimensional), or blob-like(three dimensional). These categorizations with additional information will lead to an appropriate choice of an efficient element type. The simpler element type that can be used, the less computation time will be required to attain a solution. Solid element types require the most computation time. One dimensional types require the

least.

The geometry and the loading may allow the physical situation to be categorized as a special analysis case. The cases of plane strain and plane stress are considered as possible analysis routes. These are again valuable in saving computation time if the situation lends itself to these simplifications. Finally, an appropriate element type is chosen from the set that is offered. Linear and quadratic tetrahedrals along with triangular shells and membranes are the possible alternatives for this system.

The octree method requires that the solid or surfaces to be meshed are specified. In the event of a tetrahedral mesh this means the working solid is passed to the mesher. In the event of a shell or membrane mesh the desired surfaces must be specified. **Adept** chooses an appropriate set of surfaces for the problem at hand. **Adept** also attempts to position the octree box in relation to the features that are present. At least one corner of an octree cell should be located inside the boundary of all complex features. This is automatically accomplished within the limits of the mesh density level setting.

Once the mesh has been generated the system transfers the specified loads and boundary conditions from the working solid to the now existent mesh. In the event of a surface mesh the system also associates an appropriate thickness with the surface elements. This is accomplished without user intervention. The resulting information is written out in a format that is compatible with the chosen analysis code. This is the input file for the analysis code of choice.

Once the analysis has been performed and a solution output has been generated the user is returned to the postprocessing portion of AutoSolid.FEM, in order that deformations and stress contours can be generated.

In Figure 1, the circle, Working Memory includes the facts that are asserted describing the current situation at hand. The facts that makeup the Working Memory are dynamic, and as CLIPS proceeds the resident information changes. During an **Adept** session facts are asserted and retracted to define the current status. These facts might describe characteristics of the solid being considered for finite element modeling or the facts might concern what menu is being shown to the user. The presence of facts determine what rules fire. The firing of these rules can have the following consequences. Existing facts can be retracted to remove information from the working memory. New facts can be generated. Functions can be called to obtain additional information about the modeling situation. Refer to the Programmer's Manual for a more in-depth description of the workings of the rules files and CLIPS.

In the development and testing of **Adept** there was a need to select a number of sample problems to model. Four solids were agreed upon as problems that should be used to test the system. These four test problems were worked by both the **Adept** system and practicing experts in the field of finite element analysis. Refer to Appendix C for the test solids and the mesh that resulted from an **Adept** session. The modeling results were compared to validate the fact that the techniques used by the experts and **Adept** are similar. Exact comparisons were avoided because FEA is the result of

numerous assumptions in the path of going from the real world situation to a representative model. Assumptions determine how much of the original solid is modeled and with what element types. The **Adept** system generated meshes that were similar to the choices of the practicing experts. The best and most experienced analysts often build models that do not visually match the solid of interest. **Adept** and the experienced users captured the important aspects of the real world situation. The goal is to simulate the physical world situation with the simplest model that requires the least amount of computer time to attain a solution. It is common for the resulting model used by a number of experts to not match. They may have made different assumptions in ranking the importance of some characteristics of the geometry or of the real world loading. Another interesting fact is that even for exactly the same boundary and specified loads, one should not expect two practicing analysts to generate exactly the same mesh pattern and density. There are an infinite number of meshes that will result in an acceptable answer. Again the more experienced analysts tend to choose a route that will be time efficient in creation and solution requirements.

FINAL DELIVERY VERSION OF ADEPT

The delivery version of **Adept** is capable of generating an appropriate mesh for problems requiring surface or solid element types. Many automatic meshing systems only take into account the geometry to be modeled. **Adept** attempts to tailor the meshing approach to the geometry, loads, and boundary conditions that are present. The system algorithmically attempts to determine all the needed information that it can. In many cases, before proceeding the user is given the opportunity to override the determinations of the system. The system should prove to be a good training tool in that it gives a novice user an indication of how an experienced finite element analyst proceeds from a solid to a reasonable meshing strategy. Appendix C shows some starting solids and the meshes that were generated for these problems.

Refer to the user's manual for instruction on how to use the current **Adept** system. The system does guide the user in the process of generating a reasonable mesh, so a casual user should not feel as though it is a requirement to read the entire user's manual.

The programmer's manual includes enough instruction to allow the user to make changes to the current version of **Adept**. Many examples are given on how the system works and where specific information resides. The manual explains the operation of CLIPS and the rule base in the expectation that the user may have a desire to make changes to the current version of **Adept**.

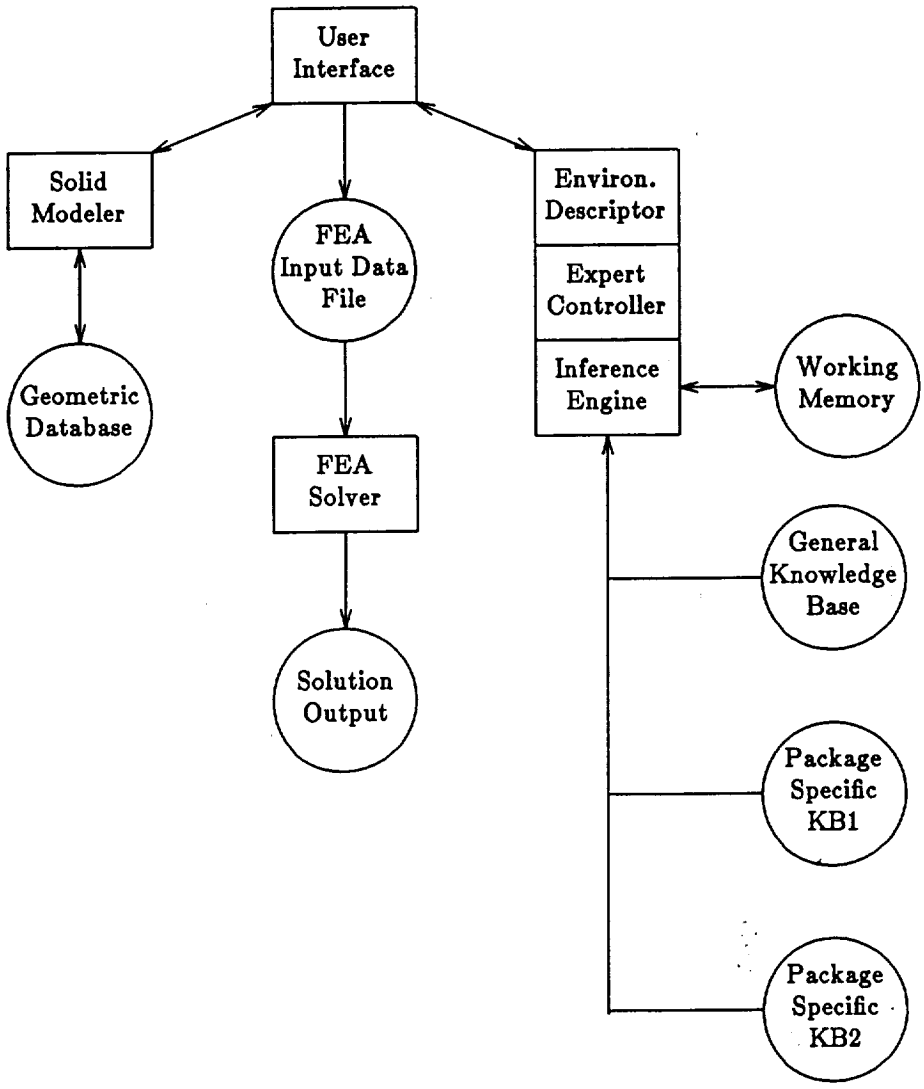


Figure 1. Architecture of Adept System

Appendix A

A Survey of Relevant CAD, FEA, and Expert System Capabilities

as of June, 1988

1. Introduction

This document describes the results of the first principal task for NASA contract #nas3-25150 (ADEPT). Ultimately, this effort will lead to the integration of an expert system containing knowledge about finite element meshing and analysis with an actual solid modeler. The resulting system should assist a design engineer in choosing a modeling and analysis strategy for a given problem. Task 1 was largely an information gathering phase of the project leading to decisions about the software environment under which the system will operate. In particular, the ADEPT expert system requires an interface to three external software systems: a computer aided design package, an expert system shell, and a finite element analysis package. The organization of this report reflects surveys done in each of these areas.

2. Computer Aided Design Environments

A key component to building the ADEPT system is to supply an effective way for a designer to describe the geometry of the part under consideration. There are several packages commercially available to provide this modeling capability. In general, these systems can be classified as either *wireframe*, *surface*, or *solid* modelers depending on the level at which they represent geometry.

Wireframe systems provide a natural extension of traditional drafting techniques. The usual primitives, lines, arcs, circles, text, etc. are allowed to be arbitrarily positioned in space. The drafting software can then project these elements according to an arbitrary view to produce mechanical drawings.

The second approach, surface modeling, extends the set of modeling primitives from one dimensional elements (lines, arcs, etc.) to surfaces. Surfaces are juxtaposed in such a manner that they (hopefully) bound a real object. Objects defined in such a manner can generally be displayed with hidden lines removed, or as a shaded solid. Because only the surface of an object is defined, however, such modelers do not have a notion of the inside or outside of an object. This precludes implementation of many important engineering applications such as three dimensional finite element meshing in such systems.

The final approach to geometric design extends the modeling building blocks from two-dimensional surfaces to solids. This approach, known as solid modeling, is becoming the technique of choice in the mechanical design industry.

A solid modeling system enables the user to create only solids which can physically exist. It creates unambiguous solids, a property essential to performing engineering applications. Finally, complex three dimensional shapes can often be constructed more quickly and efficiently than with simple drafting primitives.

Since the ultimate ADEPT system relies heavily on automatic finite element meshing facilities, a solid modeling system was deemed the most appropriate

modeling environment. Several systems were examined in order to determine a comprehensive set of features available in the industry. Included in the survey were two established solid modeling systems, as well as three newer entries into the marketplace. The results of the study are summarized in the following table.

System	Company Name	Type	Sculptured Surfaces	Hardware Platforms	Price
AutoSolid	Autodesk, Inc.	Hybrid CSG/B-rep	No	AT Comp. Workstations	5000
Concept-station	Aries	Hybrid	No custom hdwr	AT Comp	18,800
Pro/Engineer	Parametric Technology	Feature-Based	No	Workstations	9500
I/EMS	Intergraph	B-rep	Yes(NURBS)	Intergraph Workstations	~50,000
I-DEAS	SDRC	B-rep	Yes(NURBS)	Workstations	6000-50000

Table 1 - Comparison of Available Solid Modelers

2.1. AutoSolid The AutoSolid modeler marks Autodesk's entry into the MCAE market. The modeler is based on the PADL software developed at the University of Rochester during the early 1980's. The fundamental modeling elements are the quadric surfaces (plane, cone, cylinder, and sphere) plus the torus. Extensions to free form surfaces are anticipated in late 1988. Users of AutoSolid design using a combination of *sweeps* and set operations on solids (union, intersection, and difference).

An important feature of the AutoSolid environment is a C language programming library which provides outside developers access to the system capabilities through a programming interface. In particular, this allows the ADEPT development to proceed without an unduly tight coupling to the AutoSolid environment.

2.2. ConceptStation Aries Technology began marketing a solid modeler aimed at the AT level of computers in late 1986. By introducing custom hardware, they were able to perform and display set operations on solids at speeds significantly greater than those achievable through software alone. Their initial finite element support was basically a domain mapped two-d mesher, although the current release has support for automatic three-d meshing as well.

2.3. Pro/Engineer Parametric Technology introduced an innovative new solid modeler at the 1987 Autofact conference. Rather than presenting users with the general set operations for modeling, the Pro/Engineer allows a designer to work with a specific set of "features" (e.g. slots, fillets, chamfers, etc.) At present,

however, no support for finite element meshing is provided. This system currently runs in a workstation environment (SUN or Apollo).

2.4. I/EMS The "Engineering Modeling System" from Intergraph (Huntsville, AL) is an integrated environment for mechanical design and analysis. Requiring a special Intergraph workstation to run, this system is substantially more expensive than any of those previously described. I/EMS is a boundary representation modeler drawing on the *non-uniform rational b-spline* (NURBS) formalism as the basis for their geometry.

2.5. I-DEAS Like the Intergraph system, the I-DEAS package from SDRC is priced at the high end of the modeling spectrum. Although their geometry uses NURBS as the principal modeling tool, curved surfaces are approximated by planar facets for the purposes of intersection. This modeler works in conjunction with the "SUPERTAB" finite element pre and post processor.

3. Finite Element Modeling and Analysis Packages

Several of the leading finite element modeling and analysis packages were investigated for possible inclusion in the ADEPT system.

For finite element modeling purposes, surveys were conducted on the various state of the art techniques of mesh generation from a variety of papers [1,2] and consultants (Dr. Shephard, Dr. Genberg). Additionally, the handbook [3] provided an in-depth study of several of the popular analysis packages and details of particular applications. Finally, a survey of the various commercial finite element modeling capabilities has been obtained from sources such as Don Brown (D.H. Brown Associates) and Bob Johnson (CIMDATA). A brief comparison of existing state of the art systems (needed for the ADEPT system) is given in table 2.

---

1. J. W. Jones and H. H. Fong, "Evaluation of NASTRAN," *Structural Mechanics Software Series* 4 pp. 147-237 University Press of Virginia, (1984).

2. E. Schrem, "Status and Trends in Finite Element Software," *State-of-the-art surveys on Finite Element Technology* **Chapter X** pp. 325-340 The American Society of Mechanical Engineers, ().

3. C. A. Brebbia, *Finite Element Systems,* Springer-Verlag, Berlin, Heidelberg, New York (1982).

Company Name	Company Location	CAD Capabilities	FEM Capabilities	Market Emphasis
PDA	Irvine, CA	Hyperpatch	Interactive, Domain Mapped	High-end (main-framed & WS)
SDRC	Cincinnati, OH	Solidmodeling	Domain Mapped, Semi-Automatic	High-end (main-frames & WS)
ATP	Campbell, CA	Solidmodeling	Automatic frames & WS)	High-end (main-
Aries	Lowell, MA	Solidmodeling	Domain Mapped, Automatic	Low-end (PC &
Autodesk	Atlanta, GA	Solidmodeling drafting	Semi-Fully Automatic	Low-end (PC & WS)

Table 2 - Finite Element Modeling Systems

Though there are a number of other CAD packages with FEM capabilities Autodesk's package was the logical choice as the source code is easily amenable to the implementation of ADEPT algorithms and is readily available. Besides the automation of mesh generation algorithms, feature based attribute specification and complete, unambiguous geometric representation provide an ideal environment for the interaction of the expert system. Though the finite element modeling is currently restricted to solid and surface elements, the DXF interface to the industry standard AutoCAD opens the gateway for drafting and surface modeling capabilities and a variety of finite element modeling capabilities in 'lower' dimensions.

The major players in the FEA market were evaluated for product capabilities, user friendliness, service and market strengths. Apart from the various papers and articles [4] information was also gleaned from sources such as DATAQUEST (for market study). A list of some of the major players is given below.

---

1. Mark S. Shephard, Kurt R. Grice, and Marcel K. Georges, "Some Recent Advances in Automatic Mesh Generation ," *Modern Methods for Automatic Finite Element Mesh Generation*, K. Baldwin, Ed., ASCE,, (1986).

Company Name	Company Location	Pre-Processor	Solution Processor	Post-Processor	Market Emphasis
MacNeal-Schwendler	Los Angeles, CA	MSC/GRASP	MSC/NASTRAN	MSC/GRASP	General Purpose
Swanson Analysis	Houston, PA	Prep7	ANSYS	Post 12, 21	General Purpose
PDA Engineering	Irvine, CA	PATRAN-G	EASE 2	PATRAN-G	FEM, Linear Analysis
MARC analysis Corp.	Palo Alto, CA	MENTAT	MARC	MENTAT	Non-Linear
SDRC	Milford, OH	SUPERTAB	SUPERB	SUPERTAB	General Purpose
Systems Development Corp,	Minneapolis, MN		STARDYNE		General Purpose
Structural Mechanics Lab	Los Angeles, CA		SAP7		General Purpose
Pafec Eng, Cons.	Knoxville, TN		PAFEC		General Purpose
EMRC	Oak Park, MI		NISA		General Purpose
Finite Element Analysis Ltd.	London, England		LUSAS		General Purpose
Hibbit, Karlson & Sorenson	Providence, RI		ABAQUS		Non-Linear
MIT	Cambridge, MA		ADINA		Non-Linear
Georgia Inst. of Tech.	Atlanta, GA		G.T. STRUDL		Civil & Structural
Universal Analytics inc.	Los Angeles, CA		Cosmic NASTRAN		General Purpose
SRAC	Santa Monica CA		COSMOS/M		General Purpose

Table 3 - Finite Element Analysis Systems

As part of the survey we have familiarized ourselves with the capabilities of most of the above packages. In addition to obtaining and studying the manuals for MSC/NASTRAN, ANSYS, STARDYNE and MARC, we closely evaluated PATRAN, Pafec, G.T.Strudl, NISA, Cosmic NASTRAN etc.

As suggested by Henry Fong [5] the packages have been evaluated on such topics as Documentation quality (various manuals), user support, training and service facilities, element library, material library, solution options, eigenvalue extraction and time integration schemes etc. A typical comparison chart is given below.

---

5. H. H. Fong, "An evaluation of Eight U.S. General Purpose Finite Element Compute Program," *Proceedings of the 23rd AIAA/ASME/ASCE/AHS Structures, Structural Dynamics, and Materials Conference.* 1 pp. 145-160 (10-12 May 1982).

A close rapport has been established with key personnel behind some of these products and efforts were made to contact individuals who can assist us with the formulation of package specific rule base of the ADEPT system.

ANSYS and MSC/NASTRAN were the obvious choices among those surveyed. Besides their popularity in the market place (controlling more than 50% of FEA market place), the user support, the availability of expertise, and the widespread approval in aerospace and automotive industries were some of the key factors in arriving at the selection. Additionally, their stand-alone pre and post processors could be used for some of the functions not supported by AutoSolid. A decision was made to formulate the package specific rule bases for these two and extend the rule base later on to other packages. As mentioned in the monthly reports, these two analysis packages have already been obtained and installed on in-house work stations. Finally, AutoSolid, the chosen CAD and Finite Element Modeling, package has smooth interfaces to these two packages.

A few special purpose packages were also evaluated as covered in Structural Mechanics Software Series but it was decided to extend the rule base to them only in the advanced stages of the ADEPT project and thus none has been acquired. We hope to obtain some of the packages outlined in the Cosmic software catalogue (or NASA specific special purpose packages) for this purpose.

4. Expert System Considerations One of the principal goals of the task one effort was to evaluate the currently available expert system development environments. This section presents the results of this study. First, a survey of expert system shell tools and methodologies is discussed. Next, the anticipated needs for the ADEPT system are evaluated. Two packages are then discussed in greater detail. Finally, an expert system shell is chosen.

4.1. Surveys of Available Expert Systems There is currently an abundance of expert system development shells available spanning price ranges from a few hundred to tens of thousands of dollars. Two papers surveying the current market, [6, 7] were especially useful in this study. A summary of the relevant data from these papers, together with additional information from expert systems professional journals are presented in Table 4. While almost all tools offered development interfaces that aid in the debugging of the expert system, there were several significant differences among them.

Hypothetical reasoning is a characteristic not shared by all tools. This feature allows an expert system to consider multiple solutions to a given problem and determine the best solution.

---

6. C. J. Culbert, *Expert System Building Tools* , NASA Memorandum FM7(86-19), Johnson Space Center (Feb 11, 1986).

7. G. D. Riley , *Timing Tests of Expert System Building Tools* , NASA Memorandum FM7(86-51), Johnson Space Center (Apr 3, 1986).

Uncertainty management is a feature found in some but not all systems. This provides a way to assess the confidence of certain fact assertions. Furthermore, a "calculus of uncertainty" provides a mathematical means of determining the confidence associated with a conclusion based on less than certain premises.

The ability to *integrate* with other programming languages is a feature that is shared by most of the tools. This property allows calling procedures produced outside the realm of the expert system. This is especially important when dealing in an engineering environment such as the ADEPT system in which a large amount of numeric procedural computation is required.

Embeddability into application software is an important feature that allows the application to call on the expert system as a procedure.

Portability of the expert system is an important feature when embedding the system into an otherwise portable application. Many of the most robust tools are dedicated to certain hardware platforms designed for expert systems use. This greatly limits there usefulness as part of a system architecture which, for marketing reasons, needs to run on a variety of platforms.

Execution speed and the cost of the tool are the traditional measuring sticks for any software product and these two factors vary widely (Table 5).

Knowledge representation provides another significant difference among the systems. One approach is the use of rules fired based on a fact data base. These rules may be forward chaining (searching forward from established facts for a particular solution) or backward chaining (searching backwards from a known condition for the facts that may have created the condition). Another approach is the use of frames to describe object data. Frames are a collection of slots that describe the attributes of an object. Still another approach is the use of semantic nets that relate networks of objects.

The type of expert system needed by the ADEPT application will be a rule-fact base, primarily forward chaining but with some use of backward chaining. Hypothetical reasoning will be needed as well as the ability to integrate with other languages and embeddability. Portability is a major issue as the final platform or platforms will be dictated largely by market considerations. While execution speed is a concern, we do not anticipate having to make large data base searches so it is not critical. The use of frames or semantic nets do not seem appropriate. Uncertainty management does not seem to be a factor, either.

Based on these requirements, two packages have been evaluated in depth: ART by the Inference Corporation and CLIPS by NASA's Artificial Intellegence Section. These two packages have basically the same capabilities although ART certainly has a more robust development environment. Both incorporate forward and backward chaining rules. ART provides a simple syntax for both while CLIPS provides simple syntax for forward chaining rules but not for backward

chaining. Backward chaining is possible with CLIPS but one must build his own constructs to utilize this method. This is not difficult, just not as straightforward as ART.

ART provides a feature called a schema which makes use of semantic nets. This feature allows one to define a collection of facts that are interrelated. In addition, certain facts may be inherited based on other facts in the schema. Related schema may even inherit facts. This adds a bit of automation to the fact assertion process. CLIPS has no direct feature comparable to this, but it is perceived that one could be constructed. However, we do not anticipate the need for this type of feature.

Portability is clearly an advantage for CLIPS as it is written in the C programming language and will easily compile on a number of systems. ART, on the other hand, is written in LISP and is limited to a few systems although work is underway on a version written in the C language.

Cost is also an advantage for CLIPS as it is in the public domain for government work and can be obtained for a nominal cost for non-government work.

Execution speed has been an advantage for ART in the past but later revisions of CLIPS show marked improvements in this area.

Based on these findings, CLIPS is selected to be the expert system shell used for the ADEPT application.

Tool	a	b	c	d	e	f	g	h	
ART	X	X	X	X	X	X	X	LISP/C	
KEE	o	X		X	X	X	X	LISP	
KnowledgeCraft	X	o	o	X	o	X		Common LISP	
S.1		X		X	X	X		LISP/C	
ESE	o	X			X	X	X	Pascal	
KES	o	X						LISP/C	
OPS5	X				X			LISP	
OPS5+	X				X	X	X	C	
OPS83	X				X	X	X	C	
CLIPS	X	X	X		X	X	X	C	

a - Forward Chaining
b - Backward Chaining
c - Hypothetical Reasoning
d - Object Description
e - Feature Integration
f - Language Integration
g - Embeddable
h - Base Language

X - Full implementation
o - Partial implementation

Table 4 - Features of Common Expert System Shells

Tool(Version)	Platform	Seconds
ART(V2.0)	Symbolics	1.2
OPS5(VAX V2.0)	VAX	1.3
OPS5(Forgy VPS2)	Symbolics	1.7
CLIPS(V4.1)	Kaypro386	2.0
ART(V2.0)	TI Explorer	2.4
CLIPS(V2.30)	Sun	3.0
CLIPS(V4.1)	IBM AT	4.0
CLIPS(V2.30)	VAX	5.0
CLIPS(V2.30)	HP Workstation	5.0
OPS5+(V2.0003)	IBM AT	5.2
CLIPS(V2.3)	HP9000	13
OPS5+(V2.0002)	Macintosh	14
ART(V Beta 3)	VAX	17
OPS5+(V2.0003)	IBM PC	19
CLIPS(V2.10)	IBM AT	19
ExperOPS5(V1.04)	Macintosh	55
CLIPS(V2.10)	IBM PC	57
KEE(V2.1.66)	Symbolics	165

Table 5 - Relative Performance of Some Expert System Shells

Appendix B

General Knowledge Rule Base for Clips

```
; /*  CLIPS Version 4.20 4/29/88 */
; /*
; *     Copyright (C) Autodesk, Incorporated, 1990.
; *    Funded by NASA under Contract # NAS3-25150.
; */

; *      STAGE 0 RULES *
; * Get information to begin modeling session.
; * Get Solid's name, geometry, loads, bcs.
; *     [Check load balance]
; *     [Check that the bcs given are sufficient for mathematical stability]
; * Get Purpose of analysis - how detailed(big) must model be
; * Get definition of resources - max size of model(nodes, elements, or DOF)
; * There are 2 different modes used. One is Setting, the other is Showing.

; * Once clips is started the mode is chosen as setting.
; * Menu 1 is targeted as next.
(defrule Get:Started ""
        ?i-f<-(initial-fact)
=>
        (retract ?i-f)
        (assert (Menu 1))
        (assert (Setmode 0))
        (assert (Stage 0))
)

; * These are the default settings for the resources menu.
(deffacts Menu:One ""
        (Stage 0)
        (Purpose 0)
        (Hardware 0)
        (Software1 0)
        (Software2 0)
        (Time 1)
)

; * Make sure the user enters Adept with a solid.
(defrule Get:Solid ""
        (Stage 0)
        (not (Solid ? ?))
=>
        (assert (Solid =(get-solid-name) NULL))
)

; * Generate the parameters menu with the current settings.
(defrule Main:Menu ""
        (Stage 0)
        (Setmode ?switch)
        (Solid ? ?)
        ?f1<-(Menu 1)
        ?f2<-(Purpose ?purp)
        ?f3<-(Hardware ?hard)
        ?f4<-(Software1 ?soft1)
        ?f5<-(Software2 ?soft2)
        ?f6<-(Time ?time)
```

```
=>
        (bind ?ret-val (menu1-out ?purp ?hard ?soft1 ?soft2 ?time ?switch))
        (retract ?f1 ?f2 ?f3 ?f4 ?f5 ?f6)
        (assert (Purpose =(menu1-1bk)))
        (assert (Hardware =(menu1-2bk)))
        (assert (Software1 =(menu1-3bk)))
        (assert (Software2 =(menu1-4bk)))
        (assert (Time =(menu1-5bk)))
        (if (= ?switch 0)
        then
;This is the set mode
                (if (= ?ret-val -1)
                then
                        (assert (Restart))
                else
                        (assert (Menu 2))
                )
        else
;This is the check settings mode
                (if (= ?ret-val -1)
                then
                        (assert (Menu 5))
                )
        )
)

; * Characterize the solid as a stock with protrustions and depressions.
(defrule Characterize ""
        (Stage 0)
        ?f1<-(Menu 2)
=>
        (menu2-out)
        (retract ?f1)
        (assert (Menu 3))
)

; * Allow the user to apply/remove loads and boundary conditions.
(defrule Get:Forces ""
        (Stage 0)
        (Setmode ?switch)
        ?f1<-(Menu 3)
=>
        (bind ?ret-val (get-forces ?switch))
        (retract ?f1)
        (if (= ?switch 0)
        then
;This is the set mode
                (if (= ?ret-val -1)
                 then
                        (assert (Exit))
                 else
                        (assert (Loadbcs Obtained))
                )
        else
```

```
;This is the check settings mode
                (if (= ?ret-val -1)
                then
                        (assert (Loadbcs Obtained))
                )
        )
)

; * If stress is not generated by lbc set, inform the user.
(defrule Nostress:Option ""
        (Stage 0)
        (Loadbcs Obtained)
        ?f2<-(Give User Warning 1)
=>
        (retract ?f2)
        (bind ?retval (lbc-accept-option))
        (if (= ?retval 0)
        then
                (assert (Loadset Bad))
        else
                (assert (Stress Generated))
        )
)

; * In case of rigid body motion allow the user to continue if warned.
(defrule Unstable:Option ""
        (Stage 0)
        (Loadbcs Obtained)
        ?f2<-(Give User Warning 2)
=>
        (retract ?f2)
        (bind ?retval (lbc-accept-option))
        (if (= ?retval 0)
        then
                (assert (Loadset Bad))
        else
                (assert (Stability Yes))
                (assert (Direction-x Waved))
                (assert (Direction-y Waved))
                (assert (Direction-z Waved))
                (assert (Stress Generated))
        )
)

; * If the first set of lbcs is not acceptable, let them be set again.
(defrule Get:NewForces ""
        (Stage 0)
        ?f1<-(Loadbcs Obtained)
        ?f2<-(Loadset Bad)
=>
        (retract ?f1 ?f2)
        (assert (Redo Loadbcs))
)
```

```
; * Obtain the existence and sum of forces in x, y, and z.
(defrule Check:Forces ""
        (Stage 0)
        (Loadbcs Obtained)
        (not (Sigmaz ?))
=>
        (check-forces)
)

; * See if bcs are fixed or imposed displacements are in x, y, or z.
(defrule Check:Bcs ""
        (Stage 0)
        (not (Bc_limz ?))
        (Loadbcs Obtained)
=>
        (check-bcs)
)

; * If there is a nonzero force in x, y, or z then we have stress.
(defrule Generate:Stress1 ""
        (Stage 0)
        (Sigmaz ?)
        (or (Pressure Yes)
                (Forcex Yes)
                (Forcey Yes)
                (Forcez Yes))
=>
        (assert (Stress Generated))
)

; * If we didn't pass the first test then we failed test one.
(defrule Failed:Stress1 ""
        (Stage 0)
        (Sigmaz ?)
        (and (Pressure No)
                (Forcex No)
                (Forcey No)
                (Forcez No))
        (not (Stress Generated))
=>
        (assert (Strgen1 Failed))
)

; * See if boundary conditions will generate stress.
(defrule Generate:Stress2 ""
        (Stage 0)
        ?f1 <-(Strgen1 Failed)
        (Bc_limz ?)
        (or (and (Bc_limx Yes)
                        (Bc_fixx Yes))
                (and (Bc_limy Yes)
                        (Bc_fixy Yes))
                (and (Bc_limz Yes)
                        (Bc_fixz Yes)))
```

```
        (not (Stress Generated))
=>
        (retract ?f1)
        (assert (Stress Generated))
)

; * If boundary conditions don't generate stress then we failed the test.
(defrule Failed:Stress2 ""
        (Stage 0)
        ?f1 <-(Strgen1 Failed)
        (Bc_limz ?)
        (and (Bc_limx No)
                (Bc_limy No)
                (Bc_limz No))
        (not (Stress Generated))
=>
        (retract ?f1)
        (assert (Strgen2 Failed))
)

; * The fact (Stress Generated) is returned if the function check-bc-stress
; * determines that the imposed boundary conditions will be sufficient to
; * generate stress in at least one of the three directions.
; * Otherwise (Strgen2 Failed) is returned.
(defrule Generate:Stress3 ""
        (Stage 0)
        ?f1 <-(Strgen1 Failed)
        (Bc_limz ?)
        (or (and (Bc_limx Yes)
                        (Bc_fixx No))
                (and (Bc_limy Yes)
                        (Bc_fixy No))
                (and (Bc_limz Yes)
                        (Bc_fixz No)))
        (not (Stress Generated))
=>
        (retract ?f1)
        (check-bc-stress)
)

; * If no conditions exist which will generate stress then inform the user
; * that either forces or bc_limited (imposed displacements) will need to
; * be applied to have an analysis.
(defrule No:Stress ""
        (Stage 0)
        ?f2 <-(Strgen2 Failed)
=>
        (retract ?f2)
        (cl-oneline "Current Load/Bc set does not generate stress.")
        (assert (Give User Warning 1))
;       (assert (Loadset Bad))
)

; * Return the user to the loads/bcs menu
```

```
; * so that a different set of conditions will be generated.
(defrule Redo:Loadbcs ""
        (Stage 0)
        ?f1 <-(Redo Loadbcs)
        ?f2 <-(Bc_fixx ?)
        ?f3 <-(Bc_fixy ?)
        ?f4 <-(Bc_fixz ?)
        ?f5 <-(Bc_limx ?)
        ?f6 <-(Bc_limy ?)
        ?f7 <-(Bc_limz ?)
        ?f8 <-(Forcex ?)
        ?f9 <-(Forcey ?)
        ?f10 <-(Forcez ?)
        ?f11 <-(Sigmax ?)
        ?f12 <-(Sigmay ?)
        ?f13 <-(Sigmaz ?)
        ?f14 <-(Pressure ?)
=>
        (retract ?f1 ?f2 ?f3 ?f4)
        (retract ?f5 ?f6 ?f7 ?f8)
        (retract ?f9 ?f10 ?f11)
        (retract ?f12 ?f13 ?f14)
        (assert (Menu 3))
)

; * Check1 for stability in the x-direction
(defrule Check1:Directionx ""
        (Stage 0)
        (Stress Generated)
        (Bc_limz ?)
        (or (Bc_fixx Yes)
                (Bc_limx Yes))
=>
        (assert (Direction-x Stable))
)

; * Check2 for stability in the x-direction
(defrule Check2:Directionx ""
        (Stage 0)
        (Stress Generated)
        (Sigmax ?totalx)
        (and (Bc_fixx No)
                (Bc_limx No))
=>
        (bind ?small (get-epsilon))
        (if (<=  ?totalx ?small)
         then
                (assert (Direction-x Math-stability-needed))
         else
                (cl-oneline "A bc is required in the X-direction.")
                (assert (Direction-x Unstable))
        )
)
```

```
; * Check1 for stability in the y-direction
(defrule Check1:Directiony ""
        (Stage 0)
        (Stress Generated)
        (Bc_limz ?)
        (or (Bc_fixy Yes)
                (Bc_limy Yes))
=>
        (assert (Direction-y Stable))
)

; * Check2 for stability in the y-direction
(defrule Check2:Directiony ""
        (Stage 0)
        (Stress Generated)
        (Sigmay ?totaly)
        (and (Bc_fixy No)
                (Bc_limy No))
=>
        (bind ?small (get-epsilon))
        (if (<=  ?totaly ?small)
         then
                (assert (Direction-y Math-stability-needed))
         else
                (cl-oneline "A bc is required in the Y-direction.")
                (assert (Direction-y Unstable))
        )
)

; * Check1 for stability in the z-direction
(defrule Check1:Directionz ""
        (Stage 0)
        (Stress Generated)
        (Bc_limz ?)
        (or (Bc_fixz Yes)
                (Bc_limz Yes))
=>
        (assert (Direction-z Stable))
)

; * Check2 for stability in the z-direction
(defrule Check2:Directionz ""
        (Stage 0)
        (Stress Generated)
        (Sigmaz ?totalz)
        (and (Bc_fixz No)
                (Bc_limz No))
=>
        (bind ?small (get-epsilon))
        (if (<=  ?totalz ?small)
         then
                (assert (Direction-z Math-stability-needed))
         else
                (cl-oneline "A bc is required in the Z-direction.")
```

```
                (assert (Direction-z Unstable))
        )
)

; * Check for overall stability x,y and z
(defrule Stability:Overall ""
        (Stage 0)
        (Direction-x ?x)
        (Direction-y ?y)
        (Direction-z ?z)
=>
        (if (neq ?x Unstable)
         then
                (if (neq ?y Unstable)
                 then
                        (if (neq ?z Unstable)
                         then
                                (assert (Stability Yes))
                         else
                                (assert (Stability No))
                        )
                 else
                        (assert (Stability No))
                )
         else
                (assert (Stability No))
        )
)

; * If we have a case of unrestrained force then inform the user
; * to supply a different set of loads and boundary conditions.
(defrule Stability:Problems ""
        (Stage 0)
        ?f1 <-(Stability No)
        (Stress Generated)
        ?f4 <-(Direction-x ?)
        ?f5 <-(Direction-y ?)
        ?f6 <-(Direction-z ?)
=>
        (retract ?f1 )
        (retract ?f4 ?f5 ?f6)
        (cl-oneline "Current Load/Bc set allows rigid body motion.")
        (assert (Give User Warning 2))
)

; * If the lbcs are acceptable, clean up the fact list and continue.
(defrule LBCs:Acceptable ""
        (Stage 0)
        (Setmode ?switch)
        (Purpose ?)
        (Hardware ?)
        (Software1 ?)
        (Software2 ?)
        (Time ?)
```

```
        ?f1 <-(Loadbcs Obtained)
        ?f2 <-(Stress Generated)
        ?f3 <-(Stability Yes)
        ?f4 <-(Direction-x ?)
        ?f5 <-(Direction-y ?)
        ?f6 <-(Direction-z ?)
=>
        (retract ?f1 ?f2 ?f3 ?f4)
        (retract ?f5 ?f6)
        (if (= ?switch 0)
        then
                (assert (Menu 4))
        else
                (assert (Menu 5))
        )
)

; * Allow the user to rank those various regions.
(defrule Regions ""
        (Stage 0)
        (Setmode ?switch)
        ?f1<-(Menu 4)
=>
        (bind ?ret-val (get-region ?switch))
        (retract ?f1)
        (if (= ?switch 0)
        then
                (if (= ?ret-val -1)
                then
                        (assert (Redo Loadbcs))
                else
                        (assert (Menu 5))
                )
        else
                (if (= ?ret-val -1)
                then
                (assert (Menu 5))
                )
        )
)

; * Allow the user to review or alter any information in stage 1.
; * This includes parameters, resources, lbcs, and region ranking.
(defrule Show:Settings1 ""
        ?f3<-(Stage 0)
        ?f1<-(Menu 5)
        ?f2<-(Setmode ?switch)
=>
        (bind ?ret-val (menu-show-set1))
        (retract ?f1 ?f2)
        (if (= ?ret-val 0)
         then
                (assert (Setmode 1))
                (assert (Menu 1))
```

```
        )
        (if (= ?ret-val 2)
         then
                (assert (Setmode 1))
                (assert (Redo Loadbcs))
        )
        (if (= ?ret-val 3)
         then
                (assert (Setmode 1))
                (assert (Menu 4))
        )
        (if (= ?ret-val 4)
         then
                (assert (Exit))
                (retract ?f3)
        )
        (if (= ?ret-val 5)
         then
                (assert (Setmode 0))
                (assert (Menu 6))
                (retract ?f3)
        )
)

; * Categorize the geometric characteristics.
(defrule Geom:Symmetry ""
        ?f1<-(Menu 6)
        ?f2<-(Setmode ?switch)
=>
        (bind ?ret-val (get-cgeom ?switch))
        (retract ?f1 ?f2)
        (if (= ?switch 0)
        then
                (if (= ?ret-val -1)
                then
                        (assert (Setmode 1))
                        (assert (Exit))
                else
                        (assert (Setmode 0))
                        (assert (Menu 7))
                )
        else
                (if (= ?ret-val -1)
                then
                        (assert (Setmode 1))
                        (assert (Menu 8))
                )
        )
)

; * Characterize the lbc attributes as they relate to the geometric ones.
(defrule Load:Symmetry ""
        (Setmode ?switch)
        ?f1<-(Menu 7)
```

```
=>
        (bind ?ret-val (get-clbc ?switch))
        (retract ?f1)
        (if (= ?switch 0)
        then
                (if (= ?ret-val -1)
                then
                        (assert (Menu 6))
                else
                        (assert (Menu 8))
                )
        else
                (if (= ?ret-val -1)
                then
                        (assert (Menu 8))
                )
        )
)

; * Allow the user to review or alter the geometric and lbc settings.
(defrule Show:Settings2 ""
        ?f1<-(Menu 8)
        ?f2<-(Setmode ?switch)
=>
        (bind ?ret-val (menu-show-set2))
        (retract ?f1 ?f2)
        (if (= ?ret-val 0)
         then
                (assert (Setmode 1))
                (assert (Menu 6))
        )
        (if (= ?ret-val 1)
         then
                (assert (Setmode 1))
                (assert (Menu 7))
        )
        (if (= ?ret-val 2)
         then
                (assert (Exit))
        )
        (if (= ?ret-val 3)
         then
                (assert (Setmode 0))
                (assert (Determine chars))
        )
)

; * assert facts about the geometry and the lbc set.
(defrule Get:Geomchars ""
        ?f1<-(Determine chars)
=>
        (bind ?ret-val1 (get-geom-chars))
        (bind ?ret-val2 (get-lbc-chars))
        (retract ?f1)
```

```
        (assert (Reorient Loadbcs))
        (retract ?f1)
)

; * After the reorientation clear the current load bc table
; * and refill the table after any possible rotations.
(defrule Reorient:Loadbcs ""
        ?f1 <-(Reorient Loadbcs)
        ?f2 <-(Bc_fixx ?)
        ?f3 <-(Bc_fixy ?)
        ?f4 <-(Bc_fixz ?)
        ?f5 <-(Bc_limx ?)
        ?f6 <-(Bc_limy ?)
        ?f7 <-(Bc_limz ?)
        ?f8 <-(Forcex ?)
        ?f9 <-(Forcey ?)
        ?f10 <-(Forcez ?)
        ?f11 <-(Sigmax ?)
        ?f12 <-(Sigmay ?)
        ?f13 <-(Sigmaz ?)
=>
        (retract ?f1 ?f2 ?f3 ?f4)
        (retract ?f5 ?f6 ?f7 ?f8)
        (retract ?f9 ?f10 ?f11)
        (retract ?f12 ?f13)
        (check-forces)
        (check-bcs)
        (assert (Simplification Checks))
)

; * Check for plane stress and plane strain cases.  Assert
; * facts in the event these cases exist.
(defrule Plane:Stressstrain ""
        (declare (salience 35))
        (Simplification Checks)
        (or (and (Geom Extrx True)
                 (Lbc Extrx True))
                  (and (Geom Extry True)
                 (Lbc Extry True))
                  (and (Geom Extrz True)
                 (Lbc Extrz True)))
=>
        (check-strain)
)

; * This is left as an alternate way to generate plane stress/strain checks.
;(defrule Plane:Stressx      ""
;       (declare (salience 35))
;       (Simplification Checks)
;       (Geom Extrx True)
;       (Lbc Extrx True)
;       (Forcex No)
;       (Bc_fixx No)
;       (Bc_limx No)
```

```
;=>
;       (assert (Plane Stress x))
;)

;(defrule Plane:Stressy      ""
;       (declare (salience 35))
;       (Simplification Checks)
;       (Geom Extry True)
;       (Lbc Extry True)
;       (Forcey No)
;       (Bc_fixy No)
;       (Bc_limy No)
;=>
;       (assert (Plane Stress y))
;)

;(defrule Plane:Stressz      ""
;       (declare (salience 35))
;       (Simplification Checks)
;       (Geom Extrz True)
;       (Lbc Extrz True)
;       (Forcez No)
;       (Bc_fixz No)
;       (Bc_limz No)
;=>
;       (assert (Plane Stress z))
;)

;(defrule Plane:Strainx      ""
;       (declare (salience 35))
;      (Simplification Checks)
;       (Geom Extrx True)
;       (Lbc Extrx True)
;       (Bc_fixx Yes)
;=>
;        If both ends are fixed in the extrusion direction then we have
;        plane strain.
;       (check-strain)
;)

;(defrule Plane:Strainy      ""
;       (declare (salience 35))
;       (Simplification Checks)
;       (Geom Extry True)
;       (Lbc Extry True)
;       (Bc_fixy Yes)
;=>
;       (check-strain)
;)

;(defrule Plane:Strainz      ""
;       (declare (salience 35))
;       (Simplification Checks)
;       (Geom Extrz True)
```

```
;       (Lbc Extrz True)
;       (Bc_fixz Yes)
;=>
;       (check-strain)
;)

; * If geometry and lbcs mirror along x-axiz then request half of solid.
(defrule Mirror:x ""
        (declare (salience 20))
        (Simplification Checks)
        (Geom Mirx True)
        (Lbc Mirx True)
=>
        (assert (Reducex Yes))
)

; * If geometry and lbcs mirror along y-axiz then request half of solid.
(defrule Mirror:y ""
        (declare (salience 20))
        (Simplification Checks)
        (Geom Miry True)
        (Lbc Miry True)
=>
        (assert (Reducey Yes))
)

; * If geometry and lbcs mirror along z-axiz then request half of solid.
(defrule Mirror:z ""
        (declare (salience 20))
        (Simplification Checks)
        (Geom Mirz True)
        (Lbc Mirz True)
=>
        (assert (Reducez Yes))
)

; * If there is no way to reduce the direction, assert a fact.
(defrule NoMirror:x ""
        (declare (salience 10))
        (Simplification Checks)
        (not (Reducex Yes))
=>
        (assert (Reducex No))
)

; * If there is no way to reduce the direction, assert a fact.
(defrule NoMirror:y ""
        (declare (salience 10))
        (Simplification Checks)
        (not (Reducey Yes))
=>
        (assert (Reducey No))
)
```

```
; * If there is no way to reduce the direction, assert a fact.
(defrule NoMirror:z ""
        (declare (salience 10))
        (Simplification Checks)
        (not (Reducez Yes))
=>
        (assert (Reducez No))
)

; * Leave this as a way to classify the revolution cases in the
; * event of a partial revolution.
;(defrule Revolve:x ""
;       (declare (salience 20))
;      (Simplification Checks)
;       (Geom Revx True)
;       (Lbc Revx True)
;=>
;       (assert (Revolve x))
;       (degree-rev)
;)

;(defrule Revolve:y ""
;       (declare (salience 20))
;      (Simplification Checks)
;       (Geom Revy True)
;       (Lbc Revy True)
;=>
;       (assert (Revolve y))
;       (degree-rev)
;)

;(defrule Revolve:z ""
;       (declare (salience 20))
;      (Simplification Checks)
;       (Geom Revz True)
;       (Lbc Revz True)
;=>
;       (assert (Revolve z))
;       (degree-rev)
;)

; * When we are done reducing the solid proceed with choice of element type.
(defrule Simplify:over ""
        (declare (salience 5))
        ?f1<-(Simplification Checks)
=>
        (retract ?f1)
        (assert (Elemtype Checks))
)

; * For plane stress/strain use a special case element.
(defrule Elem0:Check ""
        (declare (salience 60))
        (not (Element ?))
```

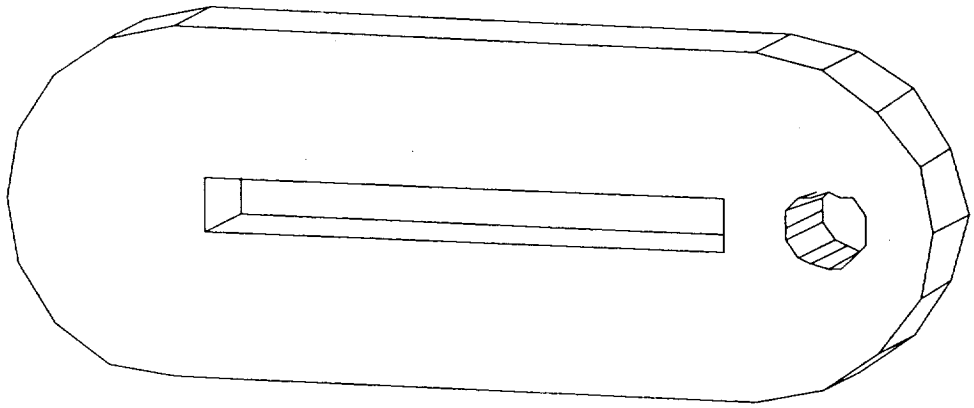
```
        (or (Plane Stress)
                (Plane Strain))
        (Elemtype Checks)
=>
        (assert (Element Membrane))
        (assert (Element Special Case))
        (assert (Reduce Solid))
)

;(defrule Elem1:Check ""
;       (declare (salience 45))
;       (not (Element ?))
;       (Elemtype Checks)
;       (Dimension twod)
;       (Lbc Parallel)
;       (Curvature Flat)
;=>
;       (assert (Element Membrane))
;       (assert (Reduce Solid))
;)

; * Use shell element type for thin shell solids.
(defrule Elem2:Check ""
        (declare (salience 45))
        (not (Element ?))
        (Elemtype Checks)
        (Dimension twod)
        (Lbc ?)
=>
        (assert (Element Shell))
        (assert (Reduce Solid))
)

; * If the solid is 3d and not a plane case then use solid element type.
(defrule Elem3:Check ""
        (declare (salience 45))
        (not (Element ?))
        (Elemtype Checks)
        (Dimension threed)
        (not (Plane Stress))
        (not (Plane Strain))
=>
        (assert (Element Solid))
        (assert (Reduce Solid))
)

; * If the element type has not been classified already, then use solid.
(defrule Elemlast:Check ""
        (declare (salience 5))
        (Elemtype Checks)
        (not (Element ?))
=>
        (assert (Element Solid))
        (assert (Reduce Solid))
```

```
)

; * If there is a direction that the solid can be reduced in then just do it.
(defrule Reduce:Solid ""
        ?f1<-(Reduce Solid)
        ?f2<-(Reducex ?x)
        ?f3<-(Reducey ?y)
        ?f4<-(Reducez ?z)
=>
        (reduce-solid ?x ?y ?z)
        (symm-bc ?x ?y ?z)
        (retract ?f1)
        (assert (Set Elemtype))
)

; * Set the element type via the api classification.
(defrule ElemSet:Clear ""
        ?f1<-(Elemtype Checks)
        ?f2<-(Set Elemtype)
        (Element ?etype)
=>
        (if (eq ?etype Solid)
         then
                (bind ?ret-val (set-elemtype "AP_EL_LTETRA"))
        )
        (if (eq ?etype Shell)
         then
                (bind ?ret-val (set-elemtype "AP_EL_LSHELL"))
        )
        (if (eq ?etype Membrane)
         then
                (bind ?ret-val (set-elemtype "AP_EL_LTRIAN"))
        )
        (if (= ?ret-val 0)
        then
                (assert (Trouble Setting Element Type))
        )
        (retract ?f1 ?f2)
        (assert (Octree Checks))
)

; * Get ready to mesh with solid element type.
(defrule Mesh:Solid ""
        (Element Solid)
        ?f1<-(Octree Checks)
=>
        (setup-mesh-solid)
        (retract ?f1)
        (assert (PreMesh Set))
)

; * Get ready to mesh with surface element type.
(defrule Mesh:Extrusion ""
        (Element Membrane)
```

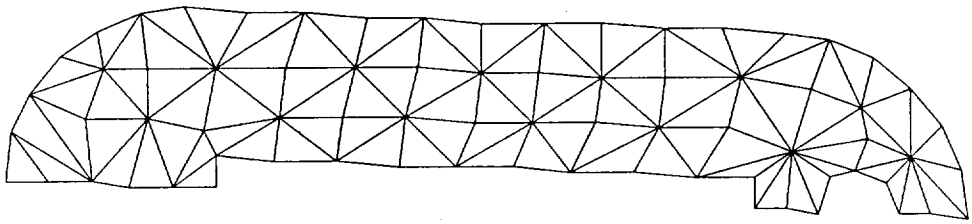
```
        (Element Special Case)
        ?f1<-(Octree Checks)
=>
        (setup-mesh-extrusion)
        (retract ?f1)
        (assert (PreMesh Set))
)

;(defrule Mesh:Planar ""
;       (Element Membrane)
;       (not (Element Special Case))
;       ?f1<-(Octree Checks)
;=>
;       (setup-mesh-planar)
;       (retract ?f1)
;       (assert (PreMesh Set))
;)

; * Get ready to mesh with surface element type.
(defrule Mesh:Shell ""
        (Element Shell)
        ?f1<-(Octree Checks)
=>
        (setup-mesh-shell)
        (retract ?f1)
        (assert (PreMesh Set))
)

; * Set all the mesh parameters and generate a mesh.
(defrule Mesh:Review ""
        ?f1<-(PreMesh Set)
=>
        (mesh-parameters)
        (retract ?f1)
        (generate-mesh)
        (assert (Mesh Exists))
)

; * When clips is over return control to api program.
(defrule Clips:Finished ""
        ?f1<-(Mesh Exists)
=>
        (retract ?f1)
        (clips-over)
)
```

Appendix C

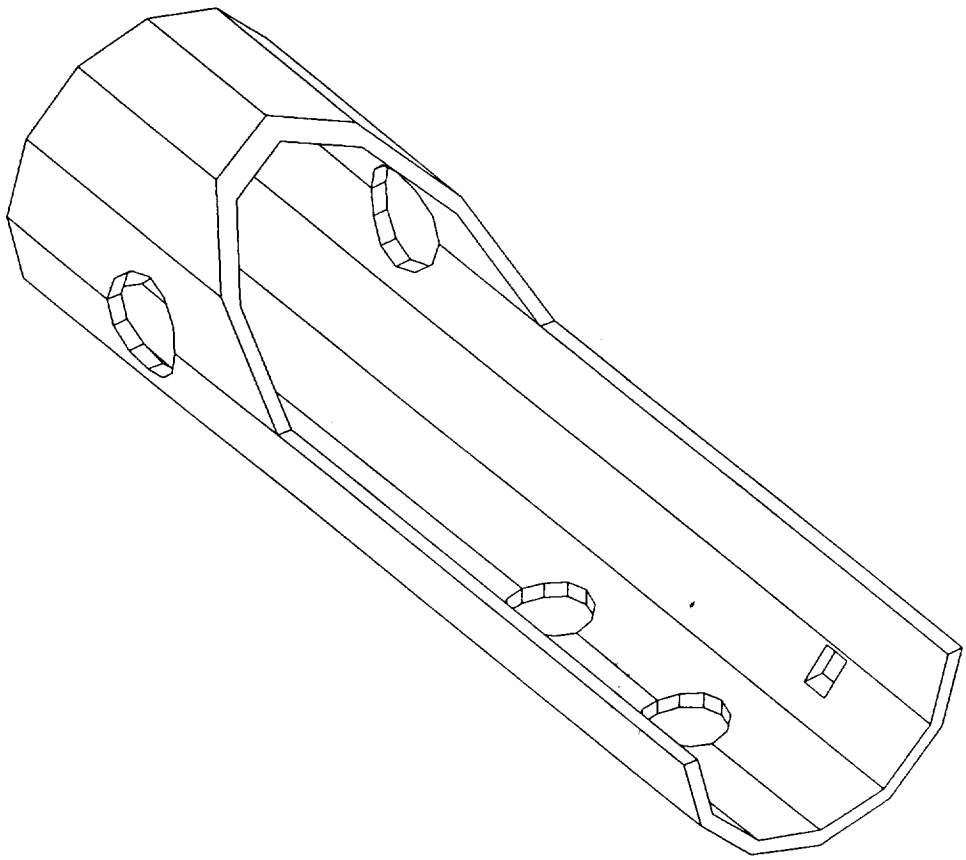
Analysis Problems with Resulting Meshes



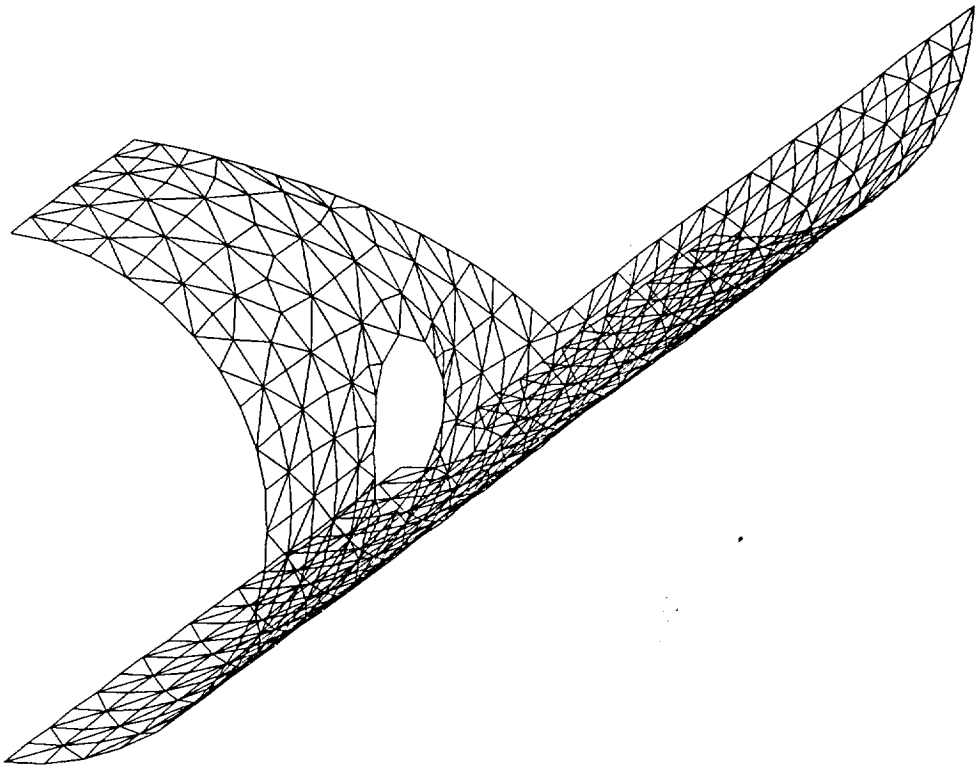
Extrusion Problem



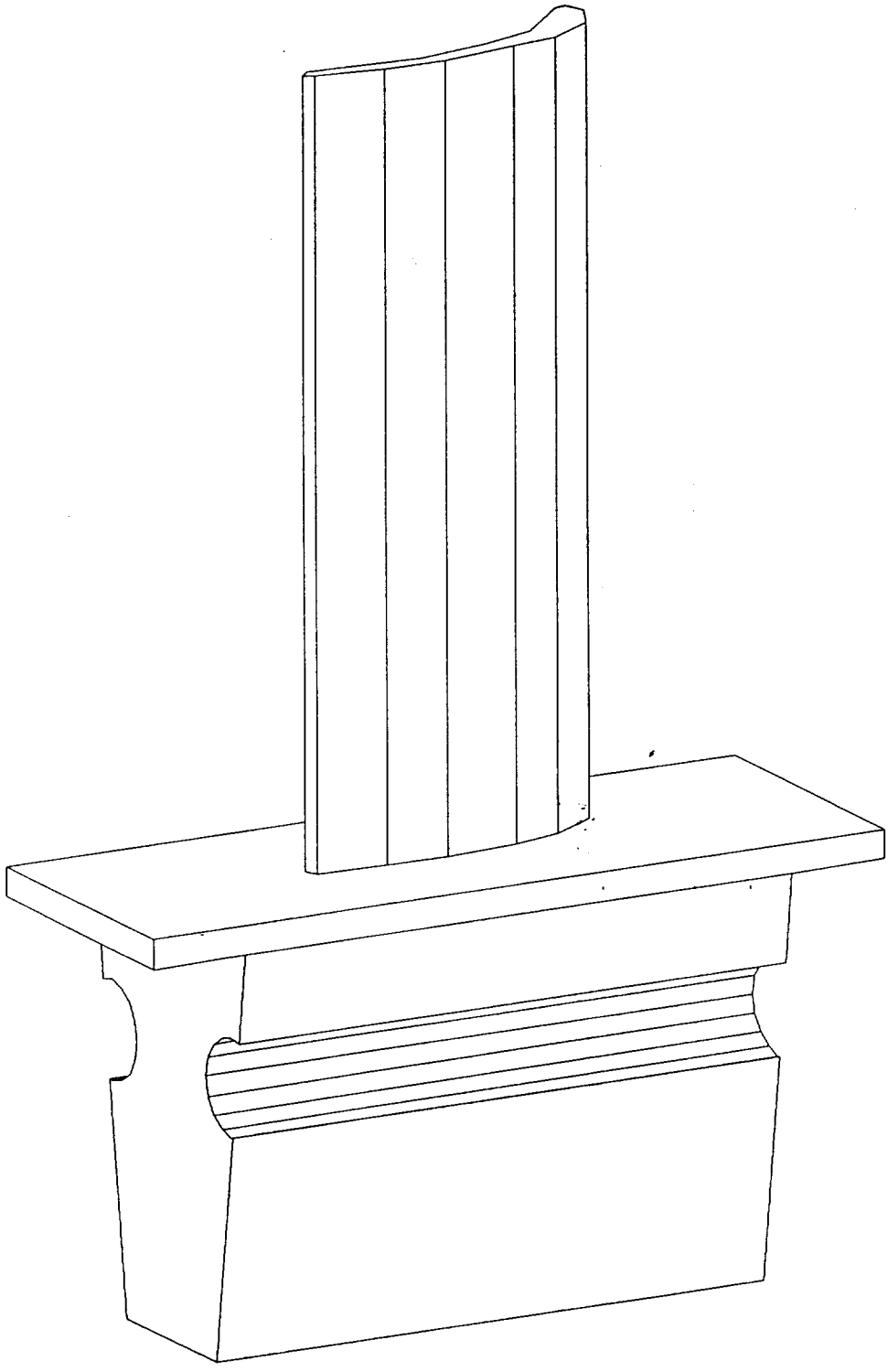
Planar Mesh with One Mirror Plane



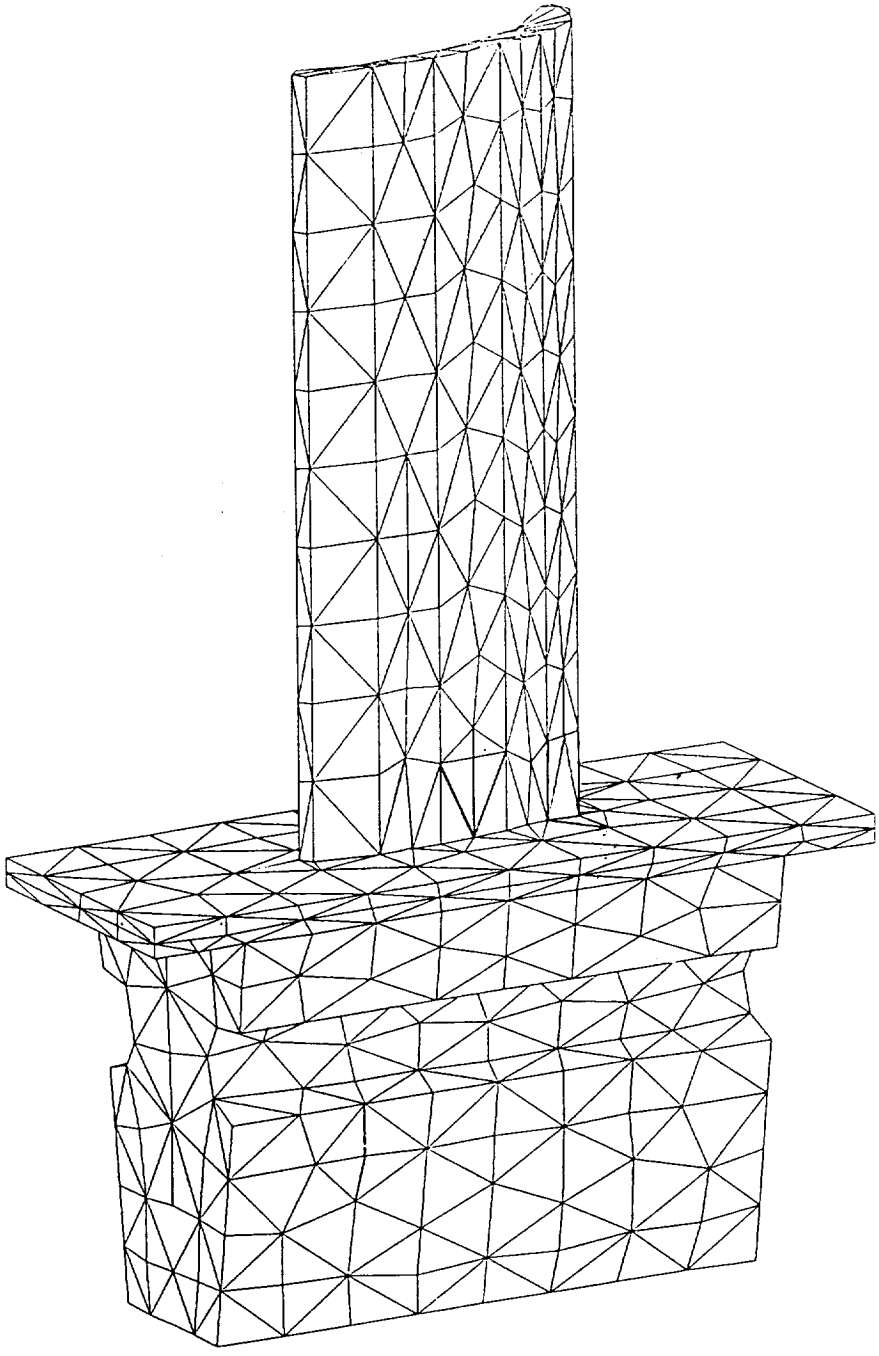
Pipe with Cutouts



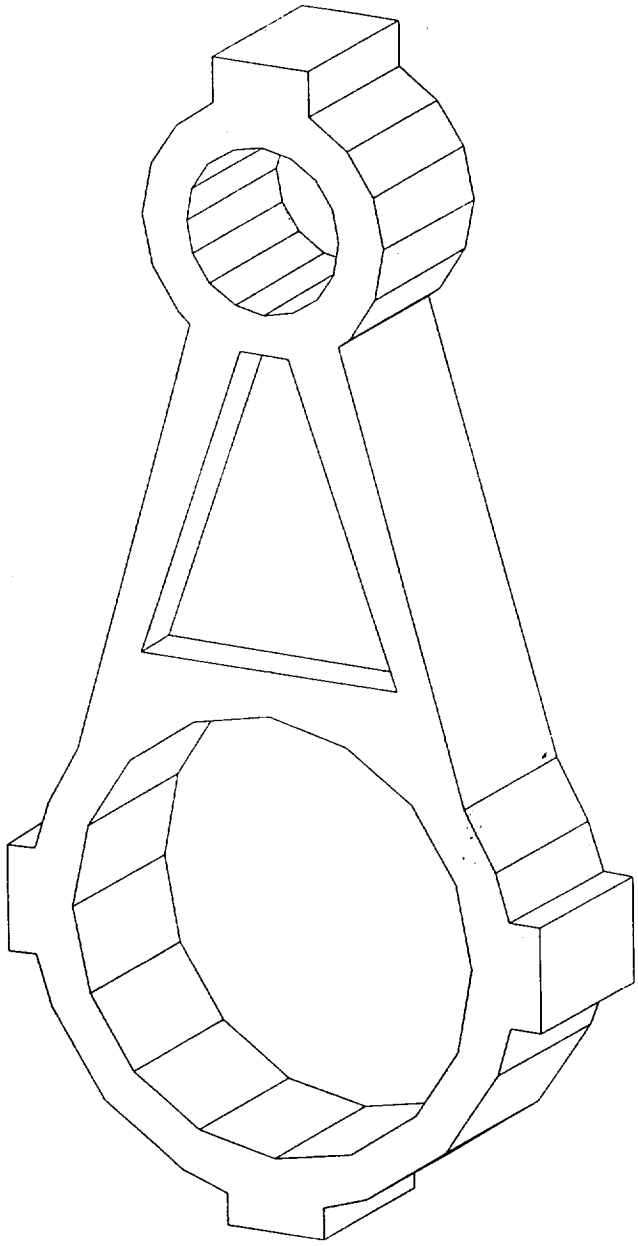
Shell Elements with One Mirror Plane and Features Filtered



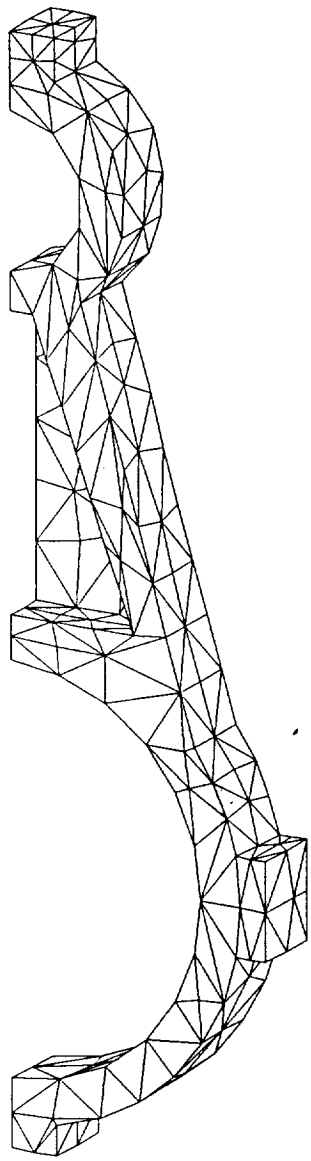
Turbine Blade Problem



Solid Elements with No Mirror Planes



Connecting Rod Problem



Solid Elements with Two Mirror Planes